Branch prediction

Branch prediction is a technique used to improve the performance of conditional branches by either analysing past branch performance through profiling—what is called *static branch prediction*—or by exploiting information about branches during program execution in order to predict one or more branch outcomes—what is called *dynamic branch prediction.*

Below, both techniques are explained in detail.

Static branch prediction

A key way to improve compile-time branch prediction is to use profile information collected from earlier runs. The key observation that makes this worthwhile is that the behaviour of branches is often bimodally distributed: an individual branch is often *highly biased* towards taken or not-taken.

Using profile information helps improve performance in a low-cost manner.

The effectiveness of any branch prediction scheme, however, depends on both the accuracy of the scheme and the frequency of conditional branches, which vary.

Dynamic branch prediction

Branch history tables

The simplest dynamic branch prediction scheme is a *branch prediction buffer* or *branch history table.*

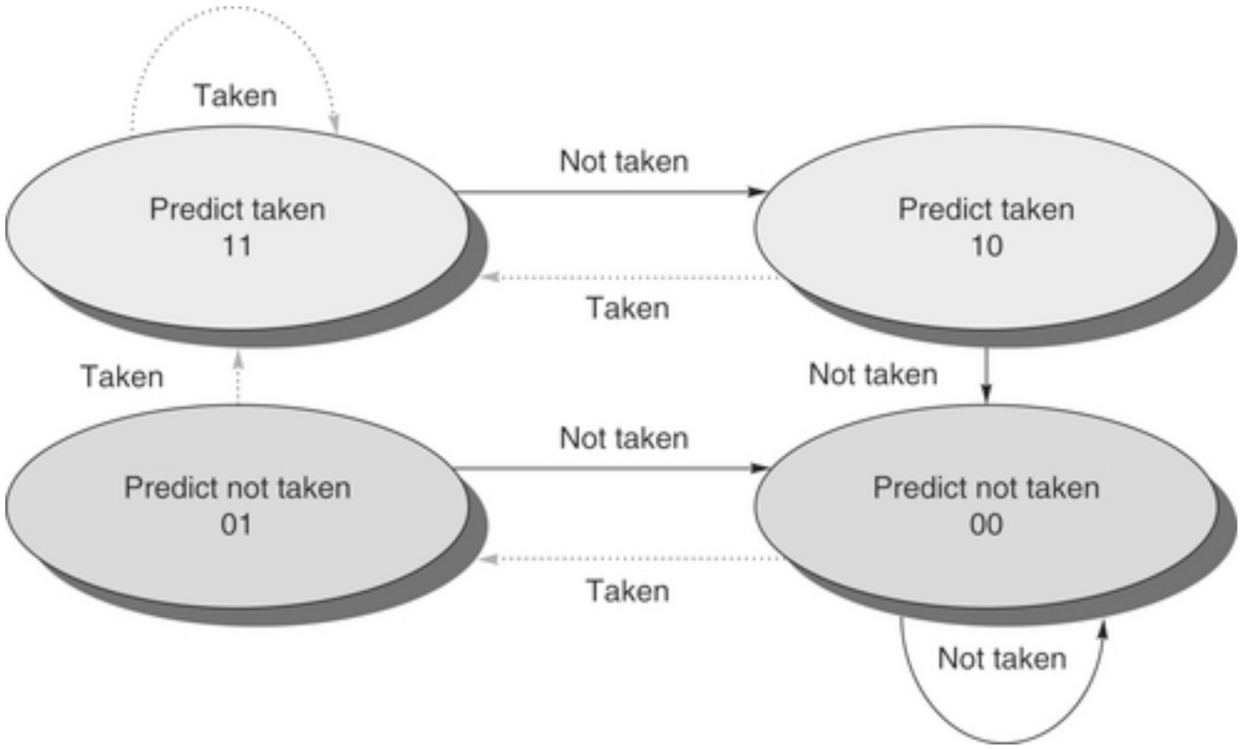
A branch prediction buffer is a small memory indexed by the lower portion of the address of the branch instruction. The memory contains a bit that says whether the branch was recently taken or not.

This scheme is the simplest sort of buffer; it has no tags and is useful only to reduce the branch delay when it is longer than the time to compute the possible target PCs.

With such a buffer, we don't know if the prediction is correct—it may have been put there by another branch that has the same low-order address bits. But this doesn't matter. The prediction is a hint that is assumed to be correct, and fetching begins in

the predicted direction. If the hint turns out to be wrong, the prediction bit is inverted and stored back.

This simple 1-bit prediction scheme has a performance shortcoming: even if a branch is almost always taken, we will likely predict incorrectly twice, rather than once, when it is not taken, since the misprediction causes the prediction bit to be flipped. To remedy this weakness, 2-bit prediction schemes are often used. In a 2-bit scheme, a prediction must miss twice before it is changed, as shown below:



Correlating branch predictors

Because of the need to enforce control dependences through branch hazards and stalls, branches will hurt pipeline performance. Loop unrolling is one way to reduce the number of branch hazards; we can also reduce the performance losses of branches by predicting how they will behave.

The 2-bit predictor schemes use only the recent behaviour of a single branch to predict the future behaviour of that branch. It may be possible to improve the prediction accuracy if we also look at the recent behaviour of other branches, rather than just the branch we are trying to predict.

Branch predictors that use the behaviour of other branches to make a prediction are called *correlating predictors* or *two-level predictors*. Existing correlating predictors add information about the behaviour of the most recent branches to decide how to predict a given branch. In the general case, an (m, n) predictor uses the behaviour of

the last m branches to choose from 2^m branch predictors, each of which is an n-bit predictor for a single branch.

The attraction of this type of correlating branch predictor is that it can yield higher prediction rates than the 2-bit scheme and requires only a trivial amount of additional hardware: the global history of the most recent m branches can be recorded in an m-bit shift register, where each bit records whether the branch was taken or not taken.